Esercitazione

Laboratorio Sistemi Operativi

Aniello Castiglione
Email: aniello.castiglione@uniparthenope.it

Esercizio

```
int glob=5;
int main() {
          pid_t pid;
          int i;
          for (i=1;i<glob;i++) {
               pid=fork();
               if (pid==0)
                    glob=glob-1;
          }
printf("Valore di glob=%d\n",glob);
}
```

Esercizio 1

- Scrivere un programma in C e Posix sotto Linux che, preso un argomento intero positivo da riga di comando, gestisca la seguente situazione:
  - genera due figli A e B e

    - se l'argomento è PARI invia un segnale SIGUSR1 alla ricezione del quale il figlio A calcola il cubo del numero passato come argomento da linea di comando, mentre il figlio B stampa un messaggio di arrivederci e termina.

    - se l'argomento è DISPARI invia un segnale SIGUSR2 alla ricezione del quale il figlio B calcola il reciproco del numero passato come argomento, attende per un numero di secondi pari al doppio del numero passato come argomento ed invia un segnale SIGUSR1 al processo A dopodiché termina l'esecuzione. Il figlio A, invece, attende la ricezione del segnale SIGUSR1, stampa un messaggio e termina.

Soluzione

```
…
//dichiarazione variabili globali
int pidA, pidB, intero, segnale;

//Gestore
void gestore(int signo)
{
segnale=signo;
}

int main (void)
{
//dichiarazione variabili locali
int cubo_int;
float rec_int;
signal(SIGUSR1, gestore);
signal(SIGUSR2, gestore);
```

Soluzione (cont.)

```
printf("Inserisci un numero intero:..\n");
scanf("%d", &intero);

//Figlio A
pidA=fork();
if(pidA==0)
{
pause();
if(segnale==SIGUSR1)//segnale SIGUSR1 inviato dal Padre
{
cubo_int=(intero*intero*intero);
printf("Figlio A: Il cubo dell'intero eseguito è: %d\n", cubo_int);
exit(0);
}
else
{
printf("Figlio A: intero DISPARI!.. Attendo un segnale da B!\n");

pause();
if(segnale==SIGUSR1) //segnale SIGUSR1 inviato da B
printf("Figlio A: Segnale arrivato!... TERMINO!!\n");
exit(0);
}
}
```

Soluzione (cont.)

```
//Figlio B
pidB=fork();
if(pidB==0)
{
pause();
if(segnale==SIGUSR1)
{
printf("Figlio B: intero PARI!... Arrivederci!!\n");
exit(0);
}
else
{
rec_int=(float)1/intero;
printf("Figlio B: reciproco intero ricevuto: %f\n", rec_int);
sleep(2*intero);
kill(pidA, SIGUSR1);
exit(0);
}
}
```

Soluzione (cont.)

```
/*Padre*/
if((pidA && pidB)!=0)
{
  sllep(1);
  if(intero%2==0)
  {
       kill(pidA, SIGUSR1);
       kill(pidB, SIGUSR1);
  }
  else
  {
       kill(pidA, SIGUSR2);
       kill(pidB, SIGUSR2);
  }
}

wait(NULL);
wait(NULL);

exit(0);
}
```

Esercizio 2

Un processo padre crea N (N numero pari) processi figli. Ciascun processo figlio Pi è identificato da una variabile intera i (i=0,1,2,3...,N-1).

Due casi:

1. Se argv[1] è uguale ad 'a' ogni processo figlio Pi con i pari manda un segnale (SIGUSR1) al processo i+1
2. Se argv[1] è uguale a 'b' ogni processo figlio Pi con i < N/2 manda un segnale (SIGUSR1) al processo i + N/2.

Soluzione

```
#include <stdio.h>
#include <ctype.h>
#include <signal.h>
#include <stdlib.h>
#define N2 5
#define N N2*2
int pg[2];
int tabpid[N];
char arg1;
```

Soluzione (cont.)

```
void handler(int signo){
  printf("Sono il processo %d e ho ricevuto il segnale
  %d\n",getpid(),signo);
}
/* Funzione eseguita da ciascun figlio: ne definisce il comportamento a
  regime */
int body_proc(int id){
  printf("Sono il processo %d con id=%d\n",getpid(),id);
  if(arg1=='a'){
/* % è l'operatore modulo, il resto della divisione intera */
  if(id % 2) pause(); /* id dispari */
  else {/* id pari */
       read(pg[0],tabpid,sizeof tabpid);
       kill(tabpid[id+1],SIGUSR1);
       }
  }
  else { /* Continua … */
```

Soluzione (cont.)

```
if(id>= N/2)
  pause();
else{
  read(pg[0],tabpid,sizeof tabpid);
  kill(tabpid[id+N/2],SIGUSR1);
  }
}
return(0);
}
int main(int argc, char* argv[])
{
int i, status;

if(argc!= 2){
 fprintf(stderr,"Uso: %s a\n o \n%s b \n",argv[0],argv[0]);
    exit(-1);}
    /* Continua … */
```

Soluzione (cont.)

```
arg1= argv[1][0];
/* primo carattere del secondo argomento
  signal(SIGUSR1,handler);
  if(pipe(pg)<0){
       perror("creazione pipe");
       exit(-1);
  }
   for(i=0;i<N;i++){
       if((tabpid[i]=fork())<0){
              perror("fork");
              exit(-1);
       }
      else if(tabpid[i]==0) {
              close(pg[1]);
              status= body_proc(i);
              close(pg[0]);
              exit(status);
       }
    }
```

Soluzione (cont.)

```
/* Il padre pone la tabella (che contiene tutti gli N
  pid dei figli) nella pipe*/
   close(pg[0]);
  printf("Sono il padre e scrivo sulla pipe le
  tabelle dei pid\n");
  for (i=1; i<=N; i++)
       write(pg[1],tabpid,sizeof(tabpid));
  close(pg[1]);
  exit(0);
}
```

Esercizio 3

Si scriva un programma in C che, utilizzando le system call di unix, preveda la seguente sintassi:

esame N N1 N2 C

dove:

esame è il nome dell'eseguibile da generare

N, N1, N2 sono interi positivi

C è il nome di un comando (presente nel PATH)

Esercizio 3 (cont.)

- Il comando dovrà funzionare nel modo seguente:
  - un processo padre P0 deve creare 2 processi figli: P1 e P2;
  - il figlio P1 deve aspettare N1 secondi e successivamente eseguire il comando C
  - il figlio P2 dopo N2 secondi dalla sua creazione dovrà provocare la terminazione del processo fratello P1 e successivamente terminare
  - nel frattempo, P2 deve periodicamente sincronizzarsi con il padre P0 (si assuma la frequenza di 1 segnale al secondo)
  - il padre P0, dopo aver creato i figli, si pone in attesa di segnali da P2: per ogni segnale ricevuto, dovrà stampare il proprio pid; all' N-esimo segnale ricevuto dovrà attendere la terminazione dei figli e successivamente terminare

Sol 3

```
#include <fcntl.h>
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
int PID1, PID2,N;
int cont=0; /* contatore dei segnali ricevuti da P2*/
void gestore_P(int sig); //gestore di SIGUSR1 per P0
void timeout(int sig); /* gestore timeout P2*/
int main(int argc , char *argv[])
{
  int N1, N2, pf, status, i;
  char com[20];
```

Sol 3

```
if (argc!=5)
{ printf("sintassi sbagliata!\n");
exit(1);
}
N=atoi(argv[1]);
N1=atoi(argv[2]);
N2=atoi(argv[3]);
strcpy(com, argv[4]);
signal(SIGUSR1, gestore_P);
PID1=fork();
```

Sol 3

```
if (PID1==0) /*codice figlio P1*/
{ sleep(N1);
  execlp(com,com,(char *)0);
  exit(0);
}
else if (PID1<0) exit(-1);
PID2=fork();
if (PID2==0)
{            /*codice figlio P2*/
  signal(SIGALRM, timeout);
  int pp=getppid();
  alarm(N2);
  for(i=1;i<=N;i++)
  { sleep(1); kill(pp, SIGUSR1);}
  exit(0);
}
else if (PID2<0) exit(-1);
```

Sol 3

```
/* padre */

while(cont<N) pause();

for (i=0; i<2; i++)
  {
   pf=wait(&status);
   if (WIFEXITED(status))
    printf("term.%d con stato %d\n",pf,WEXITSTATUS(status));
   else
    printf("term. %d inv. (segnale %d)\n",pf, WTERMSIG(status));
  }
  exit(0);
}
```

Sol 3

```
void gestore_P(int sig)
{ int i, status, pf;
  cont++;
  printf("padre %d: ricevuto %d (cont=%d)!\n", getpid(),sig, cont);
  return;
}

void timeout(int sig)
{
  printf("figlio%d: scaduto timeout!\n", getpid());
  kill(PID1, SIGKILL);
  exit(0);
}
```